Е.А. Мясникова, А.А. Тюгашев

ПАРАМЕТРИЧЕСКИЙ ГЕНЕРАТОР УПРАВЛЯЮЩИХ ПРОГРАММ РЕАЛЬНОГО ВРЕМЕНИ

(Самарский государственный аэрокосмический университет им. академика С.П. Королёва (национальный исследовательский университет))

При создании и эксплуатации современных космических аппаратов широко применяется вычислительная техника. Управление отдельными приборами и системами, входящими в комплекс бортовой аппаратуры, осуществляется специальными функциональными программами. При этом существует необходимость в некой координирующей программе, включающей в требуемые моменты времени в зависимости от ситуации на борту те или иные функциональные программы для достижения космическим аппаратом заданных целей. Такая программа и называется «*управляющей*». Разработка управляющих программ - процесс долговременный, сложный, в программах возможны ошибки и опечатки, порождающие сбои в работе бортовых систем управления, что приводит к авариям и порче дорогостоящей техники.

В связи с этим актуальна задача создания настраиваемого на необходимый язык программирования генератора управляющих программ реального времени, позволяющего улучшить процесс разработки, ускорить ход работ и сократить число ошибок.

Основой любого генератора являются три ключевых компонента: метаданные (структура, которую мы пытаемся смоделировать в программе), шаблоны (образцы по которым будет создан код) и внутренние правила, которые определяют структуру и поведение метаданных, шаблонов и их взаимодействие. На выходе генератора получаем искомый программный код. Схема генерации представлена на рисунке 1.

Рис. 1. Генерация программного кода

Метаданными является многовходовая модель управляющего алгоритма. Семантику управляющих алгоритмов реального времени (УА РВ) можно представить в виде набора четверок объектов:

$$\text{УА РВ} = \{(f_i, t_i, \tau_i, \Gamma_i)\}, \qquad i = \overline{1,N}. \tag{1}$$

где f_i – функциональная задача (программа); τ_i - момент начала выполнения действия; τ_i - длительность; $\bar{l}_i$ - логический вектор, обуславливающий действие [1]. Многовходовая модель - это набор включений управляющего алгоритма в определенные моменты времени. Ее можно представить следующей парой:

$$MWM = (W, T), \tag{2}$$

где MWM – обозначение многовходовой модели, $W = \{W_1, W_2, \ldots, W_n\}$ – множество входов управляющего алгоритма, $T = \{((W)_i, W_j), \Delta T_{ij}\}$ – отношение передачи управления, определённое на множестве W. То есть, T есть бинарное отношение на W, каждая пара входов в котором характеризуется временным интервалом ΔT_{ij}. Если, например, пара $((W)_i, W_j) \in T$, то это означает, что вход W_j запускается после выполнения входа W_i по прошествии временного интервала ΔT_{ij}. Таким образом, многовходовую модель можно представить в виде управляющего графа, где W – множество вершин, а T – множество взвешенных дуг. Пример графа многовходовой модели изображен на рисунке 2.

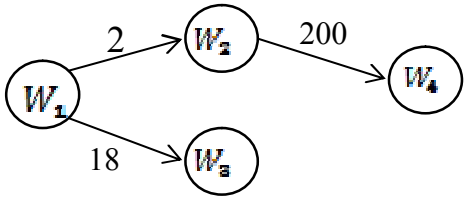

Рис. 2. Многовходовая модель

Каждый вход представляет собой логическую схему, т.е. совокупность линейных участков и ветвлений по результатам проверки логических условий.

$$W_i = \{(LU^i, Y^i)_j\}, j = \overline{1, m}, \tag{3}$$

где $LU^i = (LU_1, LU_2, \ldots, LU_n)$ – упорядоченная последовательность линейных участков, Y^i – вектор логических условий, при которых выполняется LU^i.

Каждый линейный участок можно описать в виде:

$$LU_i = \{f_k\}, f_k \in F, k = \overline{0, N}, \tag{4}$$

где $\{f_k\}$ - последовательность функциональных задач без проверок логических условий и передач управления [2].

Вход можно представить также в виде графа, где вершинами являются линейные участки, а дугами – передачи управления, которые обусловлены проверками условий логических переменных.

Пример графа входа приведен на рисунке 3.

Шаблоны генератора это образцы стандартных конструкций языка программирования, на который настраивается генератор. В программе должны присутствовать стандартные начало и завершение. Многовходовость реализуется с помощью меток и операторов безусловного перехода. Кроме этого необходим условный оператор для проверки логических переменных, описание вы-

хода из программы. К этому необходимо добавить также конструкцию, обеспечивающую необходимую задержку по времени. Итоговая структура программы представлена на рисунке 4, в блоках кода должны быть описаны последовательности запусков функциональных задач.

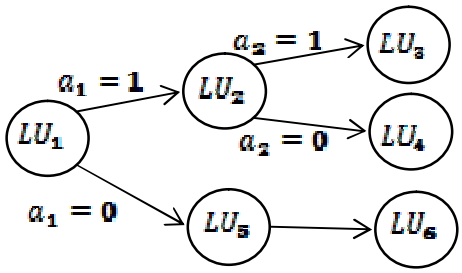

Рис. 3. Логическая схема входа

```
#include "opslib.h"
int main(int argc, char* argv[])        <-- Стандартное начало программы
{
M1:  <блок кода>
     if(<логическая переменная>)        <-- Оператор ветвления
      goto M0;
M2:  <блок кода>                        <-- Оператор безусловного перехода
      goto M1;
M0:   return 0;                         <-- Выход из программы
}                                       <-- Стандартное завершение программы
```

Метки

Рис. 4. Пример структуры программы

Если настраиваемый язык не содержит оператора безусловного перехода, то его можно заменить комбинацией переменной, заменяющей метки, цикла и переключателя.

Внутренние правила генератора - это алгоритм генерации управляющей программы. Мы начинаем генерацию с вставки стандартного начала программы и обработки первого по времени исполнения входа. Обработка входа это рекурсивное построение дерева линейных участков. Повторы исключаются вставкой команды перехода на уже описанный участок. Каждая из функциональных задач будет заменена соответствующим фрагментом кода. Некоторые функциональные задачи запрашивают на включение другие входы, исходя из этой информации, формируется очередь входов. После рассмотрения одного входа в основном алгоритме он удаляется из очереди. Повторение алгоритма продолжается до тех пор, пока очередь не опустеет. После этого мы завершаем генерируемую программу и основной алгоритм.

Таким образом, разработанный программный модуль параметрической генерации использует подход, основанный на преобразовании логико-временной схемы управляющего алгоритма над базовыми конструкциями требуемого языка программирования в программный код управляющей программы реального времени. Базовые конструкции языка программирования описы-

ваются в формате XML, логико-временная схема так же интерпретируется в XML-структуру. Это позволяет использовать технологию генерации с помощью XSLT (языка преобразования XML). XSLT имеет массу преимуществ: выполнение повторной генерации кода без перекомпиляции приложения (необходимо только изменить шаблон); возможность генерации программы на любом языке программирования; использование наглядного и широко распространенного XML формата [3].

Разработанный модуль генерации является частью программного комплекса ГРАФКОНТ, разрабатываемого по заказу ГНПРКЦ "ЦСКБ-ПРОГРЕСС" на кафедре программных систем СГАУ. Задача системы ГРАФКОНТ состоит в том, чтобы автоматизировать процессы проектирования, создания, документирования и тестирования бортовых управляющих алгоритмов и программ реального времени для космических аппаратов, обеспечить повышение качества и надежности программ, а так снизить трудоемкость и стоимость разработки.

Литература

1. Калентьев, А.А., Тюгашев, А.А. ИПИ/CALS технологии в жизненном цикле комплексных программ управления [Текст] / А.А. Калентьев, А.А. Тюгашев – Самара: Изд-во Самарского научного центра РАН, 2006. – 285 с.

2. Трусов, В.С. Система визуального конструирования временных диаграмм управляющих алгоритмов беспилотных ЛФ [Текст]: дисс. к. т. н./ Трусов Виталий Сергеевич // Самарский государственный аэрокосмический университет им. акад. С.П. Королева – Самара, 2005. – 116 с.

3. Канжелев, С.Ю. Автоматическая генерация автоматного кода [Текст] / С.Ю. Канжеле, А.А. Шалыто // Информационно-управляющие системы / Санкт-Петербургский государственный университет аэрокосмического приборостроения. – 2006. – С.35-42.

С.П. Орлов, Е.Ю. Биктимиркин, А.А. Тютнев

ИМИТАЦИОННАЯ МОДЕЛЬ МНОГОПОРТОВОЙ ПАМЯТИ МИКРОПРОЦЕССОРОВ

(Самарский государственный технический университет)

Современные микропроцессорные системы используют многопортовую (многовходовую) память для исключения конфликтов при параллельном или конвейерном выполнении потока команд. При этом возникает задача предотвращения конфликтов внутри самой многопортовой памяти, так как обращения от нескольких операционных устройств производятся к одной и той же ячейке памяти. На рис. 1 приведена схема 6-ти транзисторной ячейки SRAM памяти [1]. Здесь имеется N адресных шин B_i и N шин данных W_j , что соответствует N- входовой памяти.